REMARKS/ARGUMENTS

Summary of Rejections

Claims 1-21 were pending at the time of this Office Action.

Claims 1-6 and 19-21 stand rejected under 35 USC § 103(a) over *Klein* in view of *Bowman-Amuah*.

Claims 7-18 stand rejected under 35 USC § 103(a) over *Klein* in view of *Bowman-Amuah* and *Swartz*.

Summary of Response

The Examiner maintains his § 103(a) rejections of the claims under the same two references, namely *Klein* (U.S. 5,835,763) in view of *Bowman-Amuah* (U.S. 6,640,244), but cites new passages from *Klein* and *Bowman-Amuah* to support his rejections. In response, the Applicants point out six separate reasons why *Klein* and *Bowman-Amuah* do not teach or suggest the limitations of the independent claims.

Summary of Claims Pending

Claims 1-21 are currently pending following this response.

Applicants hereby request further examination and reconsideration of the presently claimed application.

Claim Rejections – 35 USC § 103

Claims 1-6 and 19-21 stand rejected under 35 USC § 103(a) as unpatentable over *Klein* (U.S. 5,835,763) in view of *Bowman-Amuah* (U.S. 6,640,244). Claims 7-18 stand rejected under 35 USC § 103(a) as unpatentable over *Klein* in view of *Bowman-Amuah* and *Swartz* (U.S. 6,625,651). Claims 2-21 depend on claim 1, thus claims 1-21 stand or fall on the application of *Klein* and *Bowman-Amuah* to claim 1.

Applicants respectfully submit that *Klein* and *Bowman-Amuah* do not establish a *prima facie* case of obviousness as to the pending claims. According to MPEP § 2142, three basic criteria must be met to establish a *prima facie* case of obviousness:

> First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. **Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations.** The teaching or suggestion to make the claimed combination and the reasonable expectation of success must both be found in the prior art, and not based on applicant's disclosure.

Similarly, the fact that the Examiner has the burden of proof with respect to the elements of the *prima facie* case of obviousness is also well defined in MPEP § 2142:

> The initial burden is on the examiner to provide some suggestion of the desirability of doing what the inventor has done. To support the conclusion that the claimed invention is directed to obvious subject matter, either the references must expressly or impliedly suggest the claimed invention or the examiner must present a convincing line of reasoning as to why the artisan would have found the claimed invention to have been obvious in light of the teachings of the references.

Claim 1 reads:

> 1. A method for processing a batch job, comprising:
>
> wrapping the batch job to create an application programming interface for communication with a batch framework, the batch framework comprising a batch dispatcher class, and the batch dispatcher class further comprising a method to execute the batch job; and
>
> invoking the batch framework according to a predetermined schedule via execution of a command line parameter, wherein the method provides for efficient reuse of programming code and platform independence by encapsulating the batch job and providing a uniform application programming interface for an application processing the batch job according to the method.

The Examiner has not met the burden of establishing the *prima facie* case of obviousness because neither *Klein* nor *Bowman-Amuah* teach or suggest the limitations of claim 1.

First, the Examiner cannot meet the third prong of the obviousness test because *Klein* and *Bowman-Amuah* do not teach or suggest that the batch framework comprises a batch dispatcher

class, and the batch dispatcher class comprises a method to execute the batch job. The Examiner contends that *Klein*, col. 3, lines 31-54 teaches that the batch framework comprises a batch dispatcher class, and the batch dispatcher class comprises a method to execute the batch job. The cited section of *Klein* reads:

> The thread enabling layer includes one or more application programming interfaces, or APIs, which operate to apply job scheduling requests generated by the user application for each of a series of targeted, conventional batchjobs to each such target batchjob. The API places the job scheduling request for each target batch job in the thread queue of a specialized thread job associated with that target batch job. The threadjob's task is to modify the job scheduling request in its own queue for compatibility with the target job task and place the modified request in a target queue associated with each such target batch job. It is important to note that the user application is not required to wait on the API--or on any of the batch jobs--because the user application does not require notification of completion of any such batch job or the direct return of data therefrom.
>
> When any of the target batch jobs has job completion information or data to be communicated with the user application, such information or data may immediately be placed on the common or API queue. When the user application has placed all task requests by means of the API, the user application issues a read request to the API queue and waits on the queue until awakened by data applied to the queue by one or more of target batch jobs.

As can be seen above, *Klein* teaches a single interface level between the user application and a batch job that is referred to as a thread enabling layer. Specifically, *Klein* fails to teach that his thread enabling layer is a batch framework, or that it is comprised of a batch dispatcher class further comprised of a method to execute the batch job. In contrast, claim 1 contains the limitation that the batch framework comprises a batch dispatcher class, and the batch dispatcher class comprises a method to execute the batch job. *Bowman-Amuah* is not cited to teach that the batch framework comprises a batch dispatcher class, and the batch dispatcher class comprises a method to execute the batch job, and rightfully so because *Bowman-Amuah* fails to teach such a limitation. Thus, the cited prior art fails to teach or suggest the limitation that that the batch framework comprises a batch dispatcher class, and the batch dispatcher class comprises a method

to execute the batch job. Consequently, the Examiner has failed to present a *prima facie* case of obviousness with respect to claim 1, and claim 1 should be allowed in its present form.

Second, the Examiner cannot meet the third prong of the obviousness test because *Klein* and *Bowman-Amuah* do not teach or suggest that the batch framework is invoked according to a predetermined schedule. The Examiner contends that *Klein*, col. 3, lines 31-54 teaches that the batch framework is invoked according to a predetermined schedule. The cited section of *Klein* reads:

> The thread enabling layer includes one or more application programming interfaces, or APIs, which operate to apply job scheduling requests generated by the user application for each of a series of targeted, conventional batchjobs to each such target batchjob. The API places the job scheduling request for each target batch job in the thread queue of a specialized thread job associated with that target batch job. The threadjob's task is to modify the job scheduling request in its own queue for compatibility with the target job task and place the modified request in a target queue associated with each such target batch job. It is important to note that the user application is not required to wait on the API--or on any of the batch jobs--because the user application does not require notification of completion of any such batch job or the direct return of data therefrom.
>
> When any of the target batch jobs has job completion information or data to be communicated with the user application, such information or data may immediately be placed on the common or API queue. When the user application has placed all task requests by means of the API, the user application issues a read request to the API queue and waits on the queue until awakened by data applied to the queue by one or more of target batch jobs.

As can be seen above, *Klein* teaches the processing of his batch jobs as they are applied to the queue, **not** according to a predetermined schedule. In contrast with *Klein's* teachings, claim 1 contains the limitation that that the batch framework is invoked according to a predetermined schedule. *Bowman-Amuah* is not cited to teach that that the batch framework is invoked according to a predetermined schedule, and rightfully so because *Bowman-Amuah* fails to teach such a limitation. Thus, the cited prior art fails to teach or suggest the limitation that the batch framework is invoked according to a predetermined schedule. Consequently, the Examiner has

failed to present a *prima facie* case of obviousness with respect to claim 1, and claim 1 should be allowed in its present form.

Third, the Examiner cannot meet the third prong of the obviousness test because *Klein* and *Bowman-Amuah* do not teach or suggest the limitation of using a command line parameter to invoke a batch framework. The Examiner contends that *Klein*, col. 9, lines 60-63 and col. 10, lines 25-32 teach using a command line parameter to invoke a batch framework. The cited sections of *Klein* read:

> ThreadName – This parameter specifies the name given to a thread, that is, it specifies the name of the batch job that will perform the function of the thread. This name follows the platform's standard convention.
>
> JOBQ – This particular parameter provides the name of the batch submission system that the thread job is submitted to in the format that follows the platform's standard convention.
> JOBD – This parameter provides the name of the batch job description that describes the thread job in the format that follows the platform's standard convention.

As discussed above, *Klein* does not teach or suggest a batch framework and therefore cannot teach or suggest using a command line parameter for invoking a batch framework. However, even if *Klein* taught a batch framework (and without conceding such), the referenced parameters and constants do not constitute a command line parameter. As explained by *Klein* in col. 9, lines 14-16, the ThreadName, JOBQ, and JOBD are constants and parameters used by the user's application. They are data fields that have fixed or variable values throughout the execution of a user's program. In *Klein's* disclosure, these constants and parameters are inputs but do not invoke the thread enabling layer or the batch job. Thus, *Klein* fails to teach or suggest a limitation of claim 1, namely using a command line parameter to invoke a batch framework.

Bowman-Amuah does not make up for the shortcomings of *Klein*. The Examiner contends that *Bowman-Amuah*, col. 13, lines 17-28 teaches execution of a command line

parameter. The cited section of *Bowman-Amuah* reads:

> Duplication of effort. Although class libraries allow programmers to use and reuse many small pieces of code, each programmer puts those pieces together in a different way. Two different programmers can use the same set of class libraries to write two programs that do exactly the same thing but whose internal structure (i.e., design) may be quite different, depending on hundreds of small decisions each programmer makes along the way. Inevitably, similar pieces of code end up doing similar things in slightly different ways and do not work as well together as they should.

As can be seen above, the cited section of *Bowman-Amuah* in no way relates to command line parameters or any type of execution script. Thus, the cited sections of *Bowman-Amuah* do not teach or suggest the limitation of using a command line parameter to invoke a batch framework. The Examiner attempts to resolve such a lack of teaching by further contending that *Bowman-Amuah's* batch processing system supports a UNIX platform and the UNIX platform provides execution of a command line parameter. In support of this assertion, the Examiner cites *Bowman-Amuah*, col. 96, lines 55-62. The cited section of *Bowman-Amuah* reads:

> (/Q)
> Future service delivery using /Q (usually for batch processing)
> Can prioritize messages – most important get processed sooner.
> Supports many platforms (all UNIX, NT, all common client platforms)

The above section of *Bowman-Amuah* is found under the functional description of Encina. The referenced section includes the statement that Encina "supports many platforms (all UNIX, NT, all common client platforms)." *Bowman-Amuah*, col. 96, lines 61-62. However, the cited section and the rest of the *Bowman-Amuah* specification are silent as to the use of a command line parameter for invoking a batch framework. Assuming without conceding that *Bowman-Amuah* teaches the ability of UNIX to provide execution of a command line parameter, such a teaching still does not teach its use with a batch framework, nor does it teach the contents of the command line parameter. Therefore, the ability of UNIX to execute a command line parameter and the

inclusion of a UNIX platform in *Bowman-Amuah* does not teach or suggest the limitation of invoking a batch framework via execution of a command line parameter. In contrast, claim 1 contains the limitation of using a command line parameter to invoke a batch framework. Because *Klein* and/or *Bowman-Amuah* fail to teach or suggest a limitation of claim 1, the Examiner has failed to present a *prima facie* case of obviousness with respect to claim 1, and claim 1 should be allowed in its present form.

Fourth, the Examiner cannot meet the third prong of the obviousness test because *Klein* and *Bowman-Amuah* do not teach or suggest the limitation of using a uniform API to execute the batch job. The Examiner contends that *Klein* col. 11, lines 7-11 and col. 5, lines 49-54 teach the limitation of using a uniform API to execute the batch job. The cited sections of *Klein* read:

> Applications create threads to schedule asynchronous work on its behalf. Threads execute as batch jobs and can be created into any subsystem and under any caller's authorization. Once a thread is created, other thread APIs can be used to control it.
>
> Each thread job such as thread job 25 must accept input from API 24 through thread queue 28 in a POSIX compatible format and convert or reformat the task as necessary for the target batch job so that the data provided by the target batch job is also in a POSIX compatible format.

The API of *Klein's* disclosure is not a uniform API. *Klein* teaches that it is a substantial advantage to implement the above described threaded environment simulation in non-threaded computer system in a manner which is also POSIX compatible. *See Klein*, col. 5, lines 33-36. *Klein* teaches that POSIX includes standards for asynchronous communications, which is the communication format for *Klein's* invention. *See Klein*, col. 5, lines 28-29. Therefore, *Klein* teaches that an asynchronous communication format is required to create a threaded environment for a computer system without native threading support, which is the goal of *Klein's* disclosure. As is well known in the art, non-uniform APIs have a single type of communications format,

while uniform APIs may be in either a synchronous or an asynchronous communication format. Unlike *Klein's* asynchronous API, the API of claim 1 may be in either a synchronous or an asynchronous communication format, and is thus a uniform API. Such a distinction is captured in the claim 1 limitation that the API is a uniform API. The Examiner does not contend that *Bowman-Amuah* teaches using a uniform API to execute the batch job, and rightfully so because *Bowman-Amuah* fails to teach or suggest using a uniform API to control batch jobs. Since neither *Klein* nor *Bowman-Amuah* teach or suggest using a uniform API to execute the batch job, the third prong of the obviousness test fails, and, consequently, the Examiner cannot make out a *prima facie* case of obviousness.

Fifth, the Examiner cannot meet the third prong of the obviousness test because *Klein* and *Bowman-Amuah* do not teach or suggest the limitation of using classes to execute batch jobs. The Examiner has acknowledged that *Klein* does not teach using classes to dispatch batch jobs. To resolve *Klein's* lack of teaching, the Examiner contends that *Bowman-Amuah*, col. 4, lines 30-35 teaches batch processing with classes to dispatch jobs. The cited section of *Bowman-Amuah*, part of *Bowman-Amuah's* Brief Description of the Drawings, reads:

> FIG. 55 illustrates a flowchart for a method for representing a plurality of batch jobs of a system each with a unique class in accordance with an embodiment of the present invention;
> FIG. 56 illustrates a class diagram of the batch job hierarchy;

As can be seen above, the cited section of *Bowman-Amuah* does not teach that the batch framework contains classes to dispatch the batch jobs. Rather, *Bowman-Amuah* teaches that each batch job is its own unique class. *See Bowman-Amuah*, col. 194, lines 36-38. *Bowman-Amuah* also teaches "represent[ing] each type of batch job in the system as its own class", and "adding new types of batch jobs only require[s] adding a new concrete class to the system." *Bowman-Amuah*, col. 195, lines 22-23 and 56-57. Thus, in *Bowman-Amuah's* system, the batch

jobs are unique classes within a framework. The batch jobs are compatible with the framework since they exist within the framework. **In this system, there is no use for an API since the batch jobs exist within the framework and are compatible with the framework language.** In contrast, Applicants teach a system in which a batch job is an existing program that is invoked and run using a batch framework. The Applicants' FIG. 1A illustrates this feature:

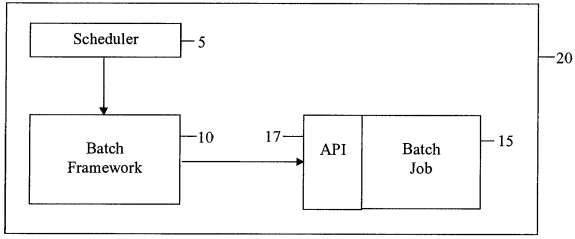


Figure 1A

As is clearly shown in Figure 1A in the Applicants' specification, the batch framework 10 and the batch jobs 15 are separated by the API 17. More specifically, the API wraps the batch jobs, thereby separating the batch jobs from the classes in the batch framework. Claim 1 specifically recites these distinctions with the limitation "wrapping the batch job to create an application programming interface for communication with a batch framework, the batch framework comprising a batch dispatcher class." Therefore, *Bowman-Amuah* fails to teach that each batch job may be a separate program and not simply a class within a framework, and teaches away from the use of existing classes to dispatch batch jobs through an API. As a result, *Bowman-Amuah* cannot teach a batch framework with classes to dispatch batch jobs outside of the framework, as taught by the Applicants. Since neither *Klein* nor *Bowman-Amuah* teach using

classes to execute batch jobs, the third prong of the obviousness test fails, and, consequently, the Examiner cannot make out a *prima facie* case of obviousness.

Sixth, the Examiner cannot meet the third prong of the obviousness test because *Klein* and *Bowman-Amuah* do not teach or suggest the limitation of encapsulating the batch job using an API. The Examiner contends that *Bowman-Amuah*, col. 12, lines 54-59 teaches encapsulating data objects. The cited section of *Bowman-Amuah* reads:

> Encapsulation enforces data abstraction through the organization of data into small, independent objects that can communicate with each other. Encapsulation protects the data in an object from accidental damage, but allows other object to interact with that data by calling the object's member functions and structures.

The cited section of *Bowman-Amuah* refers to encapsulation of data objects in an object oriented program environment. This idea is further defined as a "concept of packaging data, structures, and procedures together in one component or module." *Bowman-Amuah*, col. 10, lines 65-67. However, the Applicants use the term encapsulation differently in claim 1. The specific language of claim 1 reads "wherein the method provides for efficient reuse of programming code and platform independence by encapsulating the batch job and providing a uniform [API] for an application processing the batch job according to the method." In the context of claim 1, the encapsulation arranges the API between the batch job and the batch framework. If the definition used in *Bowman-Amuah* were applied to claim 1, then the API and the batch job would be packaged together in one component or module and would exist within the batch framework. In contrast, the Applicants teach that the API and the batch job remain individual components for communication between the batch job and the batch framework, which allows for platform independence and efficient reuse of programming code. Therefore, *Bowman-Amuah* does not teach encapsulation of the batch job, as taught by the Applicants. Since *Klein* does not teach

encapsulation and is not cited for such, neither *Klein* nor *Bowman-Amuah* teach or suggest encapsulation of the batch job. Therefore, the third prong of the obviousness test fails, and consequently, the Examiner cannot make out a *prima facie* case of obviousness. Therefore, claims 1-21 should be allowed over the cited prior art.

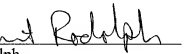
CONCLUSION

Consideration of the foregoing amendments and remarks, reconsideration of the application, and withdrawal of the rejections and objections is respectfully requested by Applicants. No new matter is introduced by way of the amendment. It is believed that each ground of rejection raised in the Office Action dated June 13, 2006 has been fully addressed. If any fee is due as a result of the filing of this paper, please appropriately charge such fee to Deposit Account Number 21-0765, Sprint. If a petition for extension of time is necessary in order for this paper to be deemed timely filed, please consider this a petition therefore.

If a telephone conference would facilitate the resolution of any issue or expedite the prosecution of the application, the Examiner is invited to telephone the undersigned at the telephone number given below.

Respectfully submitted,

Date: 8/11/06



Grant Rodolph
Reg. No. 50,487

ATTORNEY FOR APPLICANTS

CONLEY ROSE, P.C.
5700 Granite Parkway, Suite 330
Plano, Texas 75024
(972) 731-2288
(972) 731-2289 (facsimile)